Labo 2 - Protocoles applicatifs

Eléonore d'Agostino et Karim Ghozlani

05-11-2015

1. **Si une authentification par le serveur est requise, peut-on utiliser un protocole asynchrone? Quelles seraient les restrictions? Peut-on utiliser une transmission différée?**

   Oui. Par exemple, avec la classe HttpURLConnection d'Android, on peut utiliser un objet Authenticator pour définir un nom d'utilisateur et un mot de passe lors des connections HTTP. Par contre, si on n'utilise pas HTTPS, toutes ces valeurs sont envoyées au serveur sans encryption et la connexion n'est donc pas du tout sécurisée.

2. **Lors de l'utilisation de protocoles asynchrones, c'est généralement deux threads différents qui se préoccupent de l'envoi et de la réception. Quels problèmes cela peut-il poser?**

   Ceci peut poser des problèmes de concurrence. Par exemple, si on a des ressources partagées entre les deux threads, on peut se retrouver avec des valeurs aberrantes si on tente de modifier une des ressources avec les deux threads en même temps.

   Ceci peut aussi poser problème dans le cas où on chercherai à avoir une association entre une requête et sa réponse.

3. **Lorsque l'on implémente l'écriture différée, il arrive que l'on aie soudain plusieurs transmissions en attente qui deviennent possibles simultanément. Comment implémenter proprement cette situation:**

   (a) **Effectuer une connexion par transmission différée**
   C'est ce que nous avons implémenté dans le labo: une fois que l'on a de nouveau accès à internet, on envoie toutes les transmissions en attente une par une, dans l'ordre dans lequel on avait tenté de les envoyer à la base.

   (b) **Multiplexer toutes les connexions vers un même serveur en une seule connexion de transport. Dans ce dernier cas, comment implémenter le protocole applicatif, quels avantages peut-on espérer de ce multiplexage, et surtout, comment doit-on planifier les réponses du serveur lorsque ces dernières s'avèrent nécessaires?**
   Dans ce cas, une fois que l'on a de nouveau accès à internet, on envoie une seule requête contenant toutes les requêtes précédentes. Ceci nécessite donc que le serveur soit en mesure de pouvoir accepter et lire une telle requête. Enfin, il faut aussi implémenter comment multiplexer les réponses du serveur en une seule réponse, et adapter le code de notre application en conséquence.

   (c) **Comparer les deux techniques et discuter des avantages et inconvénients respectifs**
   (a) a l'avantage d'être très simple à implémenter, et ne demande pas de modifications côté serveur, mais (b) est beaucoup efficace car il ne demande qu'une seule requête, et permet donc d'éviter d'envoyer plusieurs fois les mêmes informations. Plus on a de requêtes en attente, plus (b) gagne en efficacité.

4. **Quel inconvénient y a-t-il à utiliser une infrastructure comme JSON n'offrant aucun service de validation par rapport à une infrastructure comme SOAP ou XML-RPC offrant ces possibilités? Y a-t-il en revanche des avantages que vous pouvez citer?**

   Vu qu'il n'a pas de service de validation, on ne peut pas vérifier que le contenu ou la structure du JSON est correct avant que le serveur l'ai reçu et tente d'en utiliser le contenu, et donc on ne peut pas prédire comment le serveur réagira. Alors qu'une infrastructure ayant un service de validation pourra tenter de valider le message reçu, et s'il n'est pas valide, pourra agir en conséquence.

   Par contre, JSON a l'avantage d'être plus lèger à envoyer.

5. **Quel gain peut-on espérer en moyenne sur des fichiers texte en utilisant de la compression?**

   En moyenne, on peut espérer compresser du texte par un facteur d'environ 4. Vu que le codage Base64 augmente la quantité de bytes obtenus par $1/4 * 4/3 = 1/3$, on devrait finir avec un gain d'un facteur d'environ 3.

   Par contre, on ne peut compresser que le contenu d'une requête, et non les en-têtes et de protocoles, donc pour des messages très courts, on risque de perdre plus de temps à la compression que ce qu'on aurait gagné avec le temps de transmission réduit.